II.1. Grundelemente der Programmierung

- 1. Erste Schritte
- 2. Einfache Datentypen
- 3. Anweisungen und Kontrollstrukturen
- **4. Verifikation**
- 5. Reihungen (Arrays)

4. Verifikation

- **Spezifikation: Angabe, *was* ein Programm tun soll**
  - natürliche Sprache
  - grafische Sprachen (UML, ...)
  - logische Sprachen (Z, VDM, ...)

- **Testen: Überprüfung für endlich viele Eingaben**

  ⇒ keine 100% Sicherheit

- **Verifikation: Mathematischer Beweis der Korrektheit**
  - Terminierung: Hält Programm immer an?
  - Partielle Korrektheit: Falls Programm anhält, erfüllt es Spezifikation?
  - Totale Korrektheit: Terminierung & Partielle Korrektheit

  ⇒ Semantik der Programmiersprache

Fakultät

```
public static void main (String [] arguments) {
     int n = IO.eingabe(), i, res;

     i = n;

     res = 1;

     while (i > 1) {

      res = res * i;

      i = i - 1;

     }

     System.out.println("Die Fakultät ist " + res);
 }
```

Verifikation

Programm P

```
i = n;
res = 1;
while (i > 1) {
   res = res * i;
   i = i - 1;
}
```

- **Spezifikation:**
  Programm berechnet (in **`res`**) Fakultät von **`n`**
- **Terminierung:**
  Programm hält an, weil **`i`** in jedem Schleifendurchlauf kleiner wird
- **Partielle Korrektheit:**
  Nach Ausführung ist **`res`** = **`n`**!
- **Totale Korrektheit**

Wie beweist man so etwas ?

➡ Verifikation nötig bei sicherheitskritischen Anwendungen

➡ hilft für Programmentwurf und Programmierstil

Partielle Korrektheit: Hoare-Kalkül

- **Spezifikation (zur partiellen Korrektheit)**

  $\langle \varphi \rangle$ **P** $\langle \psi \rangle$

  Wenn vor Ausführung von P Vorbedingung φ gilt

  und Ausführung von P terminiert,

  dann gilt hinterher Nachbedingung ψ.

- **Bsp:** `< true > P < res = n! >`

- Partielle Korrektheit ist *semantische* Aussage

  Hoare-Kalkül: 7 *syntaktische* Regeln zur Herleitung von Korrektheitsaussagen

Zuweisungsregel

$$\frac{}{\langle \varphi\,[x/t] \rangle\ x = t;\ \langle \varphi \rangle}$$

x ist Variable, **t** ist Ausdruck (ohne Seiteneffekte),
φ **[x/t]** ist φ mit allen **x** ersetzt durch **t**

Bsp: `<5 = 5>  x = 5;  <x = 5>`

```
<5 = 5>

x = 5;

<x = 5>
```

Konsequenzregel 1 (Stärkere Vorbedingung)

$$\frac{\langle\varphi\rangle\ P\ \langle\psi\rangle \qquad \alpha \Rightarrow \varphi}{\langle\alpha\rangle\ P\ \langle\psi\rangle}$$

Bsp: `<true>  x = 5;  <x = 5>`, denn:

$$\frac{\texttt{<5 = 5>}\ \texttt{x = 5;}\ \texttt{<x = 5>} \qquad \texttt{true} \Rightarrow \texttt{5 = 5}}{\texttt{<true>}\ \ \texttt{x = 5;}\ \ \texttt{<x = 5>}}$$

```
<true>
<5 = 5>
x = 5;
<x = 5>
```

Konsequenzregel 2 (Schwächere Nachbedg.)

$$\frac{\langle\varphi\rangle\ P\ \langle\psi\rangle \qquad \psi \Rightarrow \beta}{\langle\varphi\rangle\ P\ \langle\beta\rangle}$$

Bsp: $\langle true\rangle$ `x = 5;` $\langle x \geq 5\rangle$, denn:

$$\frac{\langle true\rangle\ x = 5;\ \langle x = 5\rangle \qquad x = 5 \Rightarrow x \geq 5}{\langle true\rangle\ x = 5;\ \langle x \geq 5\rangle}$$

$\langle true\rangle$
$\langle 5 = 5\rangle$
`x = 5;`
$\langle x = 5\rangle$
$\langle x \geq 5\rangle$

Sequenzregel

$$\frac{\langle \varphi \rangle\ P\ \langle \psi \rangle \qquad \langle \psi \rangle\ Q\ \langle \beta \rangle}{\langle \varphi \rangle\ P\ Q\ \langle \beta \rangle}$$

Bsp:

```
<true>
        x = 5;
        res = x * x + 6;
   <res = 31>
```

```
<true>
<5 = 5>

x = 5;

<x = 5>
<x * x + 6 = 31>

res = x * x + 6;

<res = 31>
```

Bedingungsregel 1

$$\frac{\langle \varphi \wedge B \rangle \ P \ \langle \psi \rangle \qquad \varphi \wedge \neg B \Rightarrow \psi}{\langle \varphi \rangle \ \texttt{if (B) \{P\}} \ \langle \psi \rangle}$$

Bsp: $\langle true \rangle$

```
        res = y;
        if (x > y) res = x;
```

$\langle res = max(x,y) \rangle$

denn: $\langle res = y \wedge x > y \rangle$

```
<x = max(x,y)>
        res = x;
```

$\langle res = max(x,y) \rangle$

und $\langle res = y \wedge \neg x > y \rangle$

$\Rightarrow \langle res = max(x,y) \rangle$

$\langle true \rangle$

$\langle y = y \rangle$

```
res = y;
```

$\langle res = y \rangle$

```
if (x > y) res = x;
```

$\langle res = max(x,y) \rangle$

Bedingungsregel 2

$$\frac{\langle \varphi \wedge B \rangle\ P\ \langle \psi \rangle \qquad \langle \varphi \wedge \neg B \rangle\ Q\ \langle \psi \rangle}{\langle \varphi \rangle\ \texttt{if (B) \{P\} else \{Q\}}\ \langle \psi \rangle}$$

Bsp: <true>

```
if (x < 0)
    res = -x;
else
    res = x;
```

<res = |x|>

denn: <x < 0>
<-x = |x|>

```
res = -x;
```

<res = |x|>

und <¬ x < 0>
<x = |x|>

```
res = x;
```

<res = |x|>

Schleifenregel

$$\frac{<\varphi \wedge B>\ P\ <\varphi>}{<\varphi>\ \texttt{while (B) \{P\}}\ <\varphi \wedge \neg B>}$$

```
<true>
      i = n; res = 1;
<i = n ∧ res = 1>
<i! * res = n!>
      while (i > 1) {res = res * i; i = i - 1; }
<i! * res = n! ∧ ¬ i > 1>
<res = n! >
```

φ ist Schleifen-invariante

denn:

```
<i! * res = n! ∧ i > 1>
<(i-1)! * (res * i) = n!>
      res = res * i;
      i = i - 1;
<i! * res = n!>
```

Hoare-Kalkül

- Zuweisungsregel

$$\frac{}{\langle \varphi\,[x/t] \rangle\ \texttt{x = t;}\ \langle \varphi \rangle}$$

- Konsequenzregeln

$$\frac{\langle \varphi \rangle\ P\ \langle \psi \rangle \qquad \alpha \Rightarrow \varphi}{\langle \alpha \rangle\ P\ \langle \psi \rangle}$$

$$\frac{\langle \varphi \rangle\ P\ \langle \psi \rangle \qquad \psi \Rightarrow \beta}{\langle \varphi \rangle\ P\ \langle \beta \rangle}$$

- Sequenzregel

$$\frac{\langle \varphi \rangle\ P\ \langle \psi \rangle \qquad \langle \psi \rangle\ Q\ \langle \beta \rangle}{\langle \varphi \rangle\ P\ Q\ \langle \beta \rangle}$$

- Bedingungsregeln

$$\frac{\langle \varphi \wedge B \rangle\ P\ \langle \psi \rangle \qquad \varphi \wedge \neg B \Rightarrow \psi}{\langle \varphi \rangle\ \texttt{if (B) \{P\}}\ \langle \psi \rangle}$$

$$\frac{\langle \varphi \wedge B \rangle\ P\ \langle \psi \rangle \qquad \langle \varphi \wedge \neg B \rangle\ Q\ \langle \psi \rangle}{\langle \varphi \rangle\ \texttt{if (B) \{P\} else \{Q\}}\ \langle \psi \rangle}$$

- Schleifenregel

$$\frac{\langle \varphi \wedge B \rangle\ P\ \langle \varphi \rangle}{\langle \varphi \rangle\ \texttt{while (B) \{P\}}\ \langle \varphi \wedge \neg B \rangle}$$

Terminierung

Für jede Schleife **`while (B) {P}`** finde einen **`int`**-Ausdruck V (*Variante* der Schleife), so dass:

$B \Rightarrow V \geq 0$ und **$\langle V = m \wedge B \rangle$ P $\langle V < m \rangle$**

```
while (i > 1) {res = res * i; i = i - 1; }
```

Variante ist `i`, denn: $i > 1 \Rightarrow i \geq 0$

```
<i = m ∧ i > 1>
<i-1 = m-1>
        res = res * i; i = i - 1;
<i = m-1>
<i < m>
```

Verifikation der Addition

```
public class Plus {

 public static void main (String [] args) {
  System.out.print ("Gib 2 Zahlen ein: ");
  int a = IO.eingabe (), b = IO.eingabe (), x, res;

  x = a;
  res = b;

  //Invariante: x ≥ 0 ∧ x + res = a + b
  //Variante: x
  while (x > 0) {
     x = x - 1;
     res = res + 1;
  }

  System.out.println (a + " + " + b + " = " + res);
}}
```

Vorbedingung: $a \geq 0$

Nachbedingung: $res = a + b$

Verifikation der Subtraktion

```
public class Subtract {

 public static void main (String [] args) {
  System.out.print ("Gib 2 Zahlen ein: ");
  int x = IO.eingabe (), y = IO.eingabe (), z, res;

  z = y;
  res = 0;

  //Invariante: x ≥ z ∧ res = z - y
  //Variante: x - z
  while (x > z) {
     z = z + 1;
     res = res + 1;
  }

  System.out.println (x + " - " + y + " = " + res);
}}
```

Vorbedingung: $x \geq y$

Nachbedingung: $res = x - y$

Verifikation eines Primzahl-Programms

```
public class Prim {
 public static void main (String [] args) {
  System.out.print ("Gib Zahl ein: ");
  int n = IO.eingabe ();
  int wurzel = (int) Math.sqrt (n),
      teiler = 2;
  boolean istPrim = true;

  //Invariante: n ≥ 2 ∧ wurzel = ⌊sqrt(n)⌋ ∧
  //        istPrim = keine Zahl i mit 2 ≤ i < teiler teilt n
  //Variante: wurzel - teiler

  while (teiler <= wurzel) {
        if (n % teiler == 0)  istPrim = false;
        teiler = teiler + 1; }

  System.out.println (n + " prim: " + istPrim);
}}
```

Vorbedingung: **n ≥ 2**

Nachbedingung: **istPrim = true** gdw. **n** ist Primzahl